**Exception** is an interrupt that cause the program to terminate when it encounters. It means it disturb the normal execution of the program. for ex;

Statement 1

Statement 2

Statement 3

Statement 4

Statement 5

Statement 6 **Exception**

Statement 7

Statement 8

Statement 9

Statement 10

In the above example the exception at the statement 6 which lead the normal execution of the program till statement 5 and terminate the next of the part.

Exception handling is a mechanism that ensure the normal execution of all the code in the program. It also allows to identify the type of exception occur.

Unchecked Exception

Checked Exc

Arithmetic Exception

Null Pointer Exception

Number Format Exception

Array Index Out of Bounds Exception

Runtime Exception

Assertion Error

JUM Error

Error

SQLException

IOException

Exception

Throwable

Object

**Exception Handling**

**A) try**

* A block that contains the code that throws the exception.
* Syntax

Try

{-------

------------

}

* Try block is used inside the methods
* Try should be followed by catch or finally.

Example

Class exception Demo {  
psvm ()

{  
try {  
 int a=10/0;

S.o.p(a);

}

Catch (exception e)

{  
S.o.p(e)

}

}

}

**Multiple catch statement**

Class Demo {  
psvm () {

Try

{

Int arr [] = {2,4,12,6}

arr [6] =10/0;

Catch (ArrayIndexOutofBoundException ae)

{

S.o.p(ae);

} Class Demo {  
psvm () {

Try

{

Int arr [] = {2,4,12,6}

arr [6] =10/0;

Catch (ArrayIndexOutofBoundException ae)

{

Catch (Arithmetic exception e);

{  
S.o.p (e);

}

Catch (Exception e)

{  
S.o.p (e)

}

}

}

**B) catch**

A block that contains the code that handle the exceptions.

* Syntax

Try{--------

---------

------------}

Catch (Exception e)

{

S.o.p(e);

}

**C) finally**

A block that contains the code that execute always whether the exception occurs or not.

* Syntax

Try{

}

Finally{ }

Or

Try{

}

Catch { }

Finally { }

}

**D) throw**

* Syntax – throw new Exception(“message”);

class vote

{  
public static void ageCheck(int age)

{

try{  
if(age<18)

throw newArithmeticException(“Not eligible for voting”);

else

S.o.p(“Go for vote”);

}  
catch (Exception e)

{  
S.o.p(e);

}

}

public static void main(String args[])

{  
try{  
ageCheck(16);

}  
catch(Exception e)

{  
S.o.p(e);

}  
}  
}

**E)throws**

* class Demo

{  
public static void main(String args[]) throws Arithmetic Exception

{  
int a=10/0;

System.out.println(a);

}

}

**ERROR**

* Error is the subclass of built in class throwable.
* Errors are the critical condition that occurs due to the lack of system resources, and it cannot be handled by the code of the program.
* Errors are caused due to catastrophic failure which usually cannot be handled by the program.
* Errors are always of unchecked type that is it occurs at the runtime environment.
* The error can be occurred if the program has error of stack overflow, out of memory error or a system crash error, these kinds of errors are due to the system. The consequences of the occurrence of error is that the program gets terminated abnormally.

**Q) Write the difference between Error and the Exception.**

**Q)** **Compare and contrast throw with throws.**

**NESTED INNER CLASS**

A class defined inside another class is called as nested class. We used a concept of inner class to logically group an interfaces in one place. So, that can be more readable and maintaineable.

**TYPES OF NESTED CLASS**

Basically there are two types of nested class :-

* Non-static nested class
* Static nested class.

Non-Static nested class can be further divided into three types:-

* Member inner class
* Method-local inner class
* Anonymous inner class.

**Member inner class**

Member inner class is a class created inside class but outside method.

Example;

class Outer {

private int data = 200;

static class Inner { // Declaring Inner class as static

public void msg() {

System.out.println("Data: " + data); // Accessing the private member variable of the Outer class

}

}

public static void sum(String[] args) { // Correcting typo and changing "Public" to "public"

Outer out = new Outer();

Outer.Inner in = new Outer.Inner(); // Creating an instance of the Inner class

in.msg(); // Correcting typo and adding semicolon

}

}

**Anonymous Inner class**

An inner class declared without a class name is called anonymous inner class. Incase of anonymous inner classes, we declare and initiate at a same time.

Example;

Abstract class person

abstract class Person {

abstract void eat();

}

class Annoy {

public static void main(String args[]) {

Person p = new Person() {

void eat() {

System.out.println("eat");

}

};

p.eat();

}

}

NOTE- A class creating for implementing interface or extended class. The java compiler decides its name “Anonymous class”.

**Method-local inner class**

In java we can write a class within a method and this will be a local type. Like local variables, the scope of inner class is restricted within the method.

Example;

class Outer {

void meth() {

int num = 20;

class Inner {

public void print() {

System.out.println(num);

}

} // end of inner class

Inner in = new Inner();

in.print();

} // end of method meth()

public static void main(String args[]) {

Outer out = new Outer();

out.meth(); // Corrected the method invocation to use lowercase 'o'

}

}

* How the java interface supports the multiple inheritance? Explain with the help of examples.

**PACKAGES IN JAVA**

Java Package is a group of similar type of classes, interfaces and sub-packages. It is a mechanism to encapsulate a group of classes, interfaces, sub-packages. It is used to provide excess protection and namespace management and to make it easier to search.

TYPES OF PACKAGES

* Built-in package – These are predefined packages in java. Ex; java.util.\*, java.io.\*
* User defined package – Java package created by user to categorized classes, interfaces, sub-packages.

**# What are the advantages of using java package.**

**CREATING PACKAGES**

Package in java can be create by using package command followed by the name of the package as the first statement of the java source file.

Example;

Package mypkg;

class Demo

{  
public static void main(String args[])

{  
System.out.println(“Hello there”);

}

}

**IMPORTING A PACKAGE**

Java have a import statements that allows us to import an entire package or use only certain class. There are three ways to access the package from outside the package.

1. Using fully qualified name.

* If we use fully qualified name then only declared class of this package will be accessible.
* There is no need to import but you need to use fully qualified name every time when you are accessing using the classes or interfaces.
* Example;

Package pack;

class Test

{  
public void show()

{  
System.out.println(“Show method”);

}  
}  
Package mypack;

Class Demo

{  
psvm()

{  
pack.Test = new pack Test():

t.show;

}

}

1. Using package name along with the class.

* We can call all the classes of a package to the other package by using the following syntax.

**Syntax**;

Import packagename.\*

Here, \* denotes all the classes and interfaces within a package named packagename.

**Example;**

Package pack;

Class Test

{  
public void show()

{  
Sout(“Show method”);

}

}  
Package mypack;

Import pack.\*;

Class Demo

{  
psvm()

{  
Test t=new Test();

t.show();

}  
}

1. Using package within particular class.

* We can call all the classes of a package to the other package by using specified class.
* Example;

Package pack;

Class Test

{  
public void show()

{  
Sout(“Show method”);

}  
}  
Package mypack;

Import pack.Test.java

Class Demo

{  
psvm()

{  
Test t=new Test();

t.show();

}  
}

**SUB-PACKAGES IN JAVA**

a